CDC-NIOSH: National Firefighter Registry Project Documentation

Team 55

Contributors: Eric Mammoser, Haolong Yan, James Davis, Joe Kurokawa, Rahul Nowlakha, and Traver Clifford

Table of Contents:

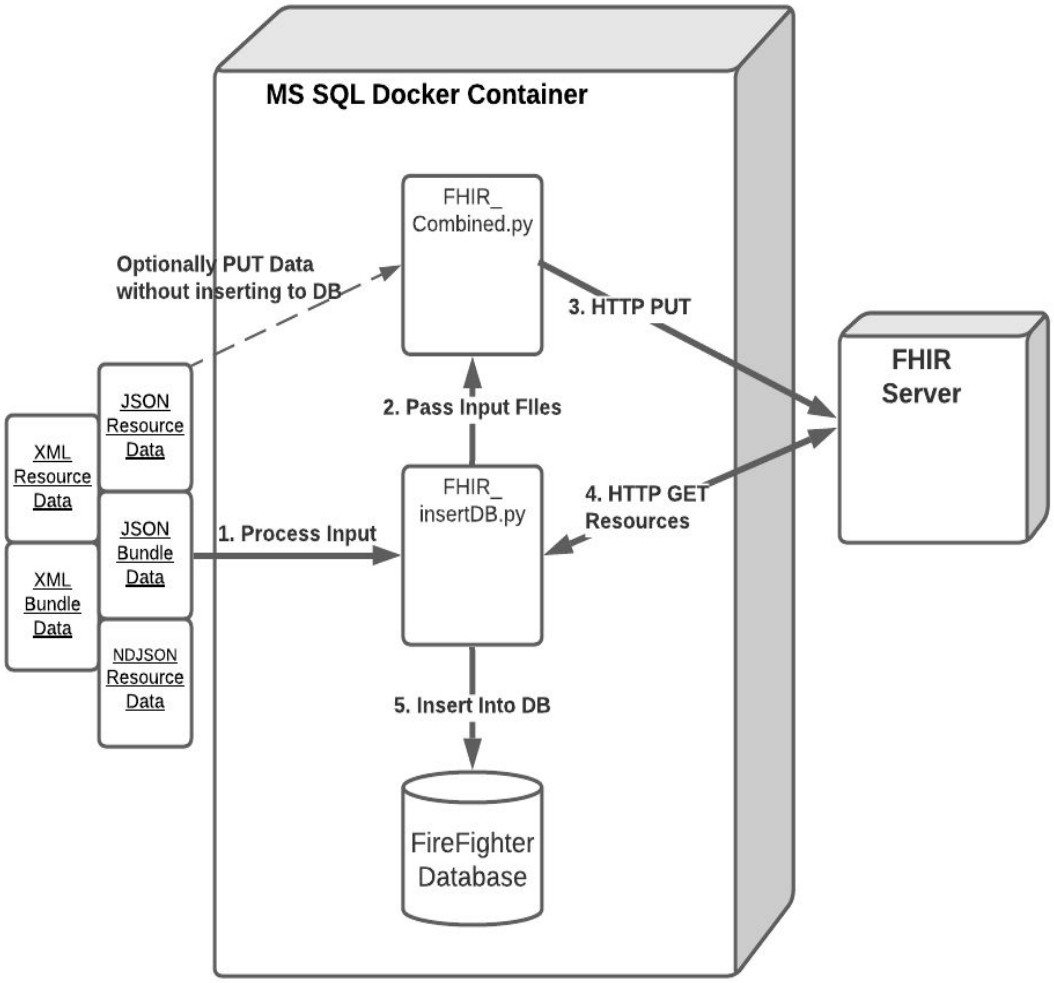
I. Overview

Studies have shown that firefighters are more susceptible to some medical conditions than the general population such as various forms of cancer, respiratory complications, and rhabdomyolysis. The National Firefighter Registry (NFR) is a CDC initiative to develop a national registry to track and analyze the long-term health of firefighters on a volunteer basis, with a specific emphasis on inclusion of women and minorities.

The goal of this project is to support this effort through the ingestion of Fire Fighter patient data provided by Fire Records Management System (FRMS) vendors. Once the data is persisted in a database, it can be used by the CDC for analytical purposes.

II. Architecture

The goal of the project is to process input files coming from Fire Records Management Systems (FRMS) and input the data into a CDC SQL Database.



Based on the diagram above, FRMS systems can send FireFighter patient and observation information in one of the following formats: XML, JSON, and NDJSON. Additionally, the XML and JSON files can include multiple patients or observations in the form of a FHIR Bundle.

Once the Input is received, the project will first verify the data is in a valid FHIR format by performing an HTTP PUT call with the information to a FHIR server. If the response that is received is successful, then the data has been verified. Next, the data is pulled from the server using an HTTP GET call. This is done so that regardless of the input format (XML, JSON, or NDJSON), the data that is used from this point on is uniform. This uniform data is then used in the database insertion portion of the application where the data will be pulled from the JSON response, mapped to the appropriate SQL table field, and then inserted into the Database.

A user can then connect the database to a Tableau dashboard to visualize the data and draw insights.

The reason for using HTTP PUT instead of POST is that for the SMARTHealthIT server which we are using, a PUT forces the creation of a resource with the ID specified in the body of the HTTP request. This is necessary for us to ensure that the Patient resource that we upload has a known ID that corresponds to the same ID as a subsequent Observation resource's subject reference ID. If we used POST and the Patient ID existed already, the patient would be created with a different ID and a subsequent invocation with an Observation resource would be linked to the existing patient and not the newly created patient.

It is also worth noting that there are Python libraries which handle FHIR client-server interaction, such as fhirclient.py, however they did not handle both JSON and XML data formats, even if the server supported it. As a result, we decided to just form our own HTTP requests so that we could directly handle both JSON and XML data formats.

III. Getting Started

In order to use the project, there are dependencies that a user must install:

1. Install the appropriate Docker version for your system
2. Clone the code repository by typing the following command in a command prompt or terminal: *git clone https://github.gatech.edu/gt-cs6440-hit-fall2020/National-Firefighter-Registry.git*
3. (Optional) To view records in the SQL database, a user can download Azure Data Studio
4. (Optional) To visualize the data, a user can download and install Tableau or another visualization software

Once a user has installed the software dependencies, the user will need to build the Docker container. To do this, the user should:

1. CD into /National-Firefighter-Registry/app
2. Type and run the following: *docker build -t nfr .*
    a. The command line parameter "nfr" can be any name that you want the container image to be labeled as.
    b. Additionally, after "nfr" there is a period stating that the "Dockerfile" that you would like to use is in the directory that the terminal is currently in.

Within the file FHIR_Combined.py, the user must change the paths to the FHIR server in the variables API_BASE and API_BUNDLE to the path of a private FHIR server. **This step is very important to prevent private patient information from being posted to a public FHIR server.**

(Optional) Within the Docker File - a user can change the password for the SQL instance by updating the following line: *SA_PASSWORD=Password!123.* However, if this is done, the user must also edit the file in

<National-Firefighter-Registry_repository>/app/create-sql-tables/setup-database.sh to reflect this new password. Additionally, the docker file will need to be rebuilt after these changes in order for the container image to be updated.

The docker build command will create a container with the following:

1. A SQL Server for Linux instance
2. A Worker, WorkerRace, WorkerObservation, and WorkerFireDepartment table
3. Python3 to run the code
4. PyoDBC to connect the Python code to the database

IV. Usage

Once the project has been built (see Section 3), a user is ready to Run the docker container and insert data into the database.

To Run the project, a user will need to Type the following command into their command prompt or terminal: *docker run -p 1433:1433 -it nfr /bin/bash*

This docker run command will start up the Docker container image with the label “nfr”. If the user decided to change the container’s name, this value would need to be updated as well. Docker will map the container’s port 1433 (the second port number) to the local machine’s port 1433 (the first port number). The first port number can be changed to whatever port you would like the host machine to use to expose the containers SQL server, however, the second port number must remain 1433 as that is the port that the SQL server within the container binds to.

The “-it” flag specifies that you would like to run the container “interactively” which allows for commands like “control+C” to be passed to the container. This is useful for cancelling or shutting down the container from within the terminal that started it.

Lastly, the command line option “/bin/bash” allows the user to enter the container in a bash shell. The user will need to start the database themselves by running “sqlfiles/entrypoint.sh”.

Once the container is running and the database has been started, a user can insert test data to verify if the code project is working:

1. Within the bash shell inside the container, type and execute: *cd test*
2. Type and execute: *./test.sh*

If production data is required instead, a user can execute the following command: *python3 FHIR_insertDB.py <FilePath to Production Data>*

There are a few restrictions when passing an input file to the script. First, the file should be either a patient resource, an observation resource, or a bundle of patient resources. We have not attempted to support a mixture of resource types here, so it is likely that it would need to be extended if that was a use model of interest.

Second, a requirement when dealing with both Patient and Observation FHIR resources is that the patient data must be read in prior to reading in the observation data. This is for two reasons: the observation data needs to reference an existing subject (patient) in the FHIR server, and the database insertion code looks for a patient with the reference ID to update that entry in the table -- if it doesn’t exist, it is not able to update the table.

At this point data will be in the database - to visualize the data, a user can connect the Tableau dashboard to the containers SQL database. To do this:

1. Within Tableau click: *Connect -> To a Server -> Microsoft SQL Server*
2. In the popup that appears, specify server as locahost,1433
3. Enter the username and password of the SQL Server instance

V. Data Validation

Data submitted from a FRMS system, will first be validated before being inserted into the tables.

To validate the data is in a valid FHIR file, the patient and observation file is POSTed to a FHIR Server, specified in FHIR_combined.py. (**Note: These URLs are to Public FHIR servers and should be changed to a private FHIR server when using production data - see section 3**)

```
smart_defaults = {
    'app_id': 'my_web_app',
    'api_base': 'https://r4.smarthealthit.org/Patient',
    'api_bundle': 'https://r4.smarthealthit.org/Bundle'
}
```

If the data provided is not in a valid FHIR format, the response from the FHIR server is logged and the data is not stored in the database.

VI. Data Extraction and Insertion

Upon receiving the response from the FHIR server, the JSON response is used to extract field values.

Data from the Patient and Observation files is mapped into specific fields in either the Worker or WorkerRace Table. Below is the how the database fields were mapped and if there are alternatives that could be mapped this is included in the Explanation column:

Database Field	Mapped FHIR Resource	Explanation
WorkerID	patient.identifier	We used this as the ID so that when trying to match observations to their respective patients we would still be able to identify the right patient
StudyCode	N/A	This is static and will always be “NFR”
GenderCode	patient.gender	As an alternative, the

		US Core Birthsex extension can be mapped. If the US Core Birthsex is available it is the better field to map to since best practice is to adhere to US Core.
EthnicityCode	US Core Ethnicity Extension	We ran into situation where there were multiple and read in the first value
RaceCode	US Core Race Extension	We read in each value to add to the WorkerRace table
StatusCode	N/A	Not mapped, there is a patient status - but it is not relevant to the NFR project
CurrentResidentialStreet	patient.valueaddress.line	These were pretty straightforward we just made sure to read the most recent address
CurrentResidentialCity	patient.valueaddress.city	Same as above
CurrentResidentialStateProv	patient.valueaddress.state	Same as above
CurrentResidentialPostalCode	patient.valueaddress.postalCode	Same as above
CurrentResidentialCountry	patient.valueaddress.country	Same as above
BirthDate	patient.birthdDate	Logic required to separate string
BirthMonth	patient.birthdDate	Logic required to separate string
BirthDay	patient.birthdDate	Logic required to separate string
Birthyear	patient.birthdDate	Logic required to separate string
BirthPlaceCountry	patient-birthPlace extension	This extension was the only one we found

		that makes sense
BirthPlaceCity	patient-birthPlace extension	Same as above
BirthPlaceStateProv	patient-birthPlace extension	Same as above
LastObservedDate	N/A	Use Observation.lastUpdated
LastObservedMonth	N/A	Use Observation.lastUpdated
LastObservedDay	N/A	Use Observation.lastUpdated
LastObservedyear	N/A	Use Observation.lastUpdated
SSN	US SSN Identifier	Was straightforward from the FHIR standard
LastName	patient.name.family	If multiple that contained use we used official
FirstName	patient.name.Given	If multiple that contained use we used official
MiddleName	patient.name.Given	If multiple that contained use we used official
LastNameAlias	Same as LastName but where use = nickname	Only populated if name with use as nickname
FirstNameAlias	Same as FirstName but where use = nickname	Only populated if name with use as nickname
MiddleNameAlias	Same as MiddleName but where use = nickname	Only populated if name with use as nickname
Site	N/A	The best location for

		this would be Organization.name
DiagnosedWithCancer	Binary Value	If an Observations with loinc code 21861-0 is present, then 1.
DiagnosedWithCancerStateProv	N/A	This will require the encounter bundle.
PrimaryEmailAddress	Patient.telecom.value	Check if system is Email
OptInEmails	N/A	This may not readily available in FHIR data
MobilePhoneNumber	Patient.telecom.value	Check if system is Phone
OptInTextMessages	N/A	This may not readily available in FHIR data
Comments	N/A	Observation.note may be a good field for this
IdmsID	Null	We were told there was not a value we needed to map here
ImportCode	N/A	We were told there was not a value we needed to map here
SourceFile	File Path to Input File	Populated with the patient file that was provided
LastUpdateDate	Date of last Update	Updated at each update to patient info
LastUpdatedBy	Admin	Static Value
CreateDate	Date the Patient record is created	Use SSN as unique identifier
CreatedBy	Admin	Static Value